Automatically Utilizing Science Opportunities with Continuous Plan Improvements

Gregg Rabideau, Barbara Engelhardt, Steve Chien
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA

ABSTRACT

We describe a methodology for representing and optimizing user preferences on plans. Our approach differs from previous work on plan optimization in that we employ a generalization of commonly occurring plan quality metrics, providing an expressive preference language. We introduce a domain independent algorithm for incrementally improving the quality of feasible plans with respect to preferences described in this language. Finally, we experimentally show that plan quality can be significantly increased with very little modeling effort for each domain.

KEYWORDS: planning, scheduling, optimization, plan quality, spacecraft operations

INTRODUCTION

Traditionally, AI planning has been mainly concerned with generating feasible plans that satisfy a set of goals. In many domains, it is insufficient to simply model the hard constraints of the system. Many undesirable, feasible solutions may exist which satisfy the goals. In addition, strict feasibility constraints may be too weak for most problems, but necessary for completeness. For example, while it may be physically possible to completely drain a battery, reasons of risk and longevity will make it preferable to maintain a certain level of charge. However, this preferred charge level, if encoded as a hard constraint, would preclude solutions where a full battery drain was necessary. We need to be able to evaluate plan variables at a finer granularity than simply as being consistent or violated. To achieve this, we build on the traditional representation of discrete *hard constraints* and mandatory goals to include continuous *soft constraints* (i.e., preferences) and optional goals. In other words, we extend the notion of what *must* be accomplished (and how) to what *should* be accomplished (and how). In this way, the user can specify which feasible solutions are more desirable, establishing a basis for automatically generating high quality plans.

In many NASA domains, the user can have a complicated definition of plan quality. For example, scientists typically would like to complete as many experiments as possible within given windows of opportunity. Other users, such as engineers, might have a preference for fewer power switches of a spacecraft instrument in order to extend the life of the instrument. Certain system states may be more desirable than other states. For example, extending an arm of a rover might leave the rover unstable, making it preferable to keep the arm stowed when not in use. Some timing constraints may be flexible but also

have a preferred time. For example, a calibration may be most useful immediately before an experiment, but still have some utility up to five minutes earlier. As another example, an experiment may have several different ways of collecting data, each resulting with different levels of data quality. We present a general representation of plan quality that is capable of encoding a wide range of preferences including the ones just described.

We implement our representation of plan quality in the ASPEN planning and scheduling system (Fukunaga et al. 1997; Rabideau et al. 1999). In addition, we demonstrate our approach to plan optimization using a generalization of a technique called *iterative repair* (Minton and Johnston 1988; Zweben et al. 1994). During repair, the conflicts in the schedule are detected and addressed one at a time until no conflicts exist, or a user-defined time limit has been exceeded. A conflict is a violation of a plan constraint, and can be resolved by making certain modifications to the plan. The most common plan modifications include moving an activity, adding a new instance of an activity, and deleting an activity. For each conflict, a domain-independent repair expert automatically generates modifications that could potentially repair the conflict.

During *iterative optimization*, low scoring preferences are detected and addressed individually until the maximum score is attained, or a user-defined time limit has been exceeded. A preference is a quality metric for a plan variable, and can be improved by making modifications to the plan similar to repair. For each preference, a domain-independent improvement expert automatically generates modifications that could potentially improve the preference score.

The iterative optimization algorithm has many of the same desirable properties as iterative repair. Both algorithms can be invoked at any time on any plan, making them more amenable to mixed-initiative planning. Repairing or improving existing plans enables fast turn-around times when small changes create conflicts or degrade plan quality. Changes may occur from manual modifications or from automatically detecting unexpected differences during execution. Further discussions with applications to spacecraft commanding can be found in (Chien et al. 1998).

In the next section, we describe the ASPEN planning model. Then, we describe an extension to this model for representing plan quality. Next, we present one possible algorithm for optimization that uses this representation. The next section discusses how we view optimization in the overall planning and execution architecture. Finally, we introduce a realistic NASA domain with complicated quality metrics for which we have easily encoded preferences and quickly improved plan quality during execution.

THE ASPEN PLANNING MODEL

In ASPEN, we have adopted a planning model with an explicit representation of constraints for time, resources and states (Smith et al. 1998). Plan operators, called activities, have a set of local variables including a start time and duration. Activities may have a set of temporal constraint variables, each specifying a minimum and maximum separation between two activities in the plan. Activities also share a number of global resources or state variables. Local constraint variables may be defined in an activity specifying the required value of a resource or state variable for the activity. The combined effects of the activities define the time-varying profiles (i.e., timelines) for the values of the resources and state variables. Global constraints can be defined for each timeline, restricting its set of legal values. For resources, these are capacity constraints.

For state variables, the set of legal states and transitions can be specified. The ASPEN planning model also includes a representation for activity hierarchies. Activities can have a disjunctive set of decompositions, each of which expands the activity into different set of sub-activities. A local variable represents the currently selected decomposition. Arbitrary functional relationships can be expressed between any of the variables in the activities. This allows ASPEN to make external calls to special reasoning modules for calculating plan values, if necessary.

Finally, ASPEN has an explicit representation of mandatory and optional goals. Goals are simply activity specifications that do not immediately appear in the plan. A mandatory goal is a conflict until the activity has been inserted into the plan (i.e., the goal is satisfied). Optional goals are not considered conflicts but instead degrade plan quality when not satisfied.

REPRESENTING PLAN QUALITY

We define preferences as quality metrics for variables in complete plans. Preferences provide a mechanism for specifying which plan variables are important to plan quality. Certain values of these variables are preferred over others, without regard for legality. We define a set of preference classes that directly corresponds to the set of plan variable classes.

Preference Variables

To better understand what types of preferences are included in our semantics, we must describe the types of plan variables that can contribute to plan quality. There are five basic types: local activity variable, activity/goal count, resource/state variable, resource/state change count, state duration.

An activity variable preference indicates a ranking for the values of a local variable in an activity instance in the plan. Local activity variables include domain-specific variables as well as internal variables for start time, end time, duration, resource usage, temporal distance from other activities, and selected decomposition. Typically, a preference is made for variables with a particular name defined in a particular type of activity. For example, minimizing tardiness in (Williamson and Hanks 1994; Miyashita and Sycara 1995) is a preference on the end times of activities that fulfill factory orders. Minimizing work in process (WIP) is a preference on the distance between the order request and order fulfillment activities. Other preferences can score the plan based on the number of existing activities of specific types (i.e., activity count). Or, one can make a general preference for satisfying more of the optional goals. In a typical spacecraft domain, scientists prefer to include as many observation activities as possible in a limited window of opportunity.

A preference can also be made for certain values of a global resource or state variable. A resource/state variable preference ranks the set of resource/state values that exist within the planning horizon. For example, a preference can be made for maximizing the minimum value of a battery over time. Other preferences can score the plan based on the number changes occurring on a resource/state variable (i.e., resource/state change count). This type of preference could be used to limit the number of power spikes on the battery. Finally, a preference can be made on the duration of a particular state on a state variable.

Pointing a spacecraft antenna towards earth, for example, is preferred when the spacecraft is not constrained to any other state.

Mapping to Quality Metrics

A preference is a mapping from a plan variable to a quality metric (i.e., score) in the interval [0,1] (see Figure 1). Specifically, a preference indicates whether the score is monotonically increasing or decreasing with respect to the plan variable within certain bounds. The user can also specify that the score increases as the difference with a given fixed value decreases. In other words, the high score is centered on a value selected from the domain of the variable. From this high-level specification, mapping functions are generated that take preference variables as arguments and return real-valued scores.

Each preference includes an upper and lower bound to indicate the range of the variable for which the score increases or decreases. Any values outside this range produce a score of either zero or one. For example, anything over 90% battery charge may be indistinguishable in terms of quality. Therefore, a preference can be defined as increasing with minimum charge and reaching a maximum score at 90% charge. Each preference also includes a weight for specifying the relative importance of the preference to overall plan quality. The score of a plan is computed as the weighted average of scores for plan variables with preferences.

An aggregate preference is defined for many plan variables, and can either score each variable independently, or score the result of applying a function to the variables. If the preference scores each variable, then the scores are weighted equally and averaged. The built-in functions that can be used in aggregate preferences include average, sum, minimum, and maximum. These functions constitute the set of functions most commonly observed in preferences from various domains. For example, minimizing makespan is a preference on the maximum end time of all activities in the schedule. The specified function is computed for the current set of plan variables, and the result is mapped to a score for the preference.

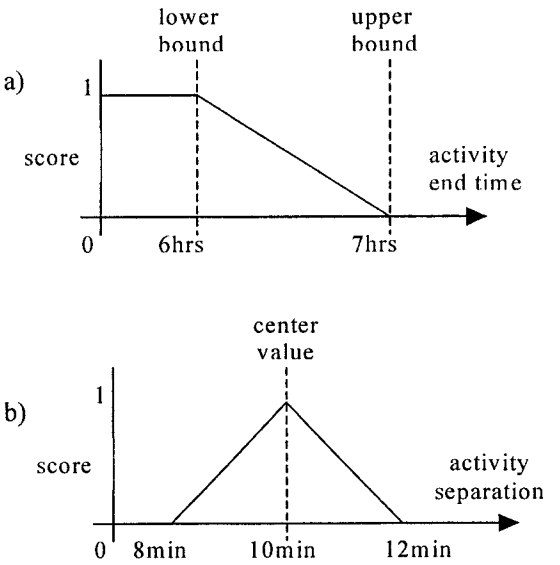

Figure 1: a) Mapping the end time of an activity to a score. This implements a preference for minimizing tardiness of an activity. The deadline is at the sixth hour and the score decreases to zero one hour after the deadline. b) Mapping the distance between two activities to a score centered on a given value. This implements a preference for maintaining a 10 minute separation with a ±2 minute tolerance.

IMPROVING PLAN QUALITY

Preferences allow us to define quality metrics for evaluating feasible plans and making quantitative distinctions between different plans. The next step is to use these preferences to generate high quality plans.

Local Improvement Experts

In addition to establishing quality metrics, preferences can provide insight into how to improve plan quality. We define a domain-independent *improvement expert* for each class of preference to aid in optimization (see Figure 2). The expert uses the preference specification to find plan modifications that will improve the score for the given preference and current plan. In other words, an expert is a link between changes in the plan and the change in quality. For example, if less resource usage were preferred, expert improvements would include deleting an activity that is currently using the resource. It is a local expert, however, and does not guarantee an increase in overall plan quality. Improvement experts provide a framework for optimization algorithms, defining the search space of possible improvements. We define a separate class of improvement expert for each class of preference.

Local activity variable expert. First, one class of expert is used for improving preferences on local activity variables. The most obvious modification for improving this preference is to change the value of the local variable. The expert only considers variables that are currently contributing to the low score. For example, only the end time of activity a2 in Figure 2 can be changed to improve the score for this preference. If score is a decreasing function of the variable, then making an improvement requires assigning a value less than its current value. Likewise, we must assign a value greater than its current value to improve an increasing preference. In cases where the variable is the start or end time of the activity, assigning a value implies moving the activity to earlier or later times. Expert modifications also include creating activities with high scoring values or deleting activities with low scoring values for the specified variable.

Activity/goal count expert. A different class of improvement expert is used preferences on the number of activities/goals. For a given preference of this class, there is only one expert modification. When the preference is for more occurrences of a goal/activity, creating new activities is the only beneficial modification. When the preference is for fewer occurrences, deleting existing activities is the only improvement.

Resource/state variable expert. Another class of expert improves preference scores for the values of resources or state variables. Only activities that use the resource or state variable are considered. For a high resource preference, the expert selects activities that increase the resource when adding and activities that decrease the resource when deleting. Just the opposite is true for low resource preferences. When moving, if the preference is for a

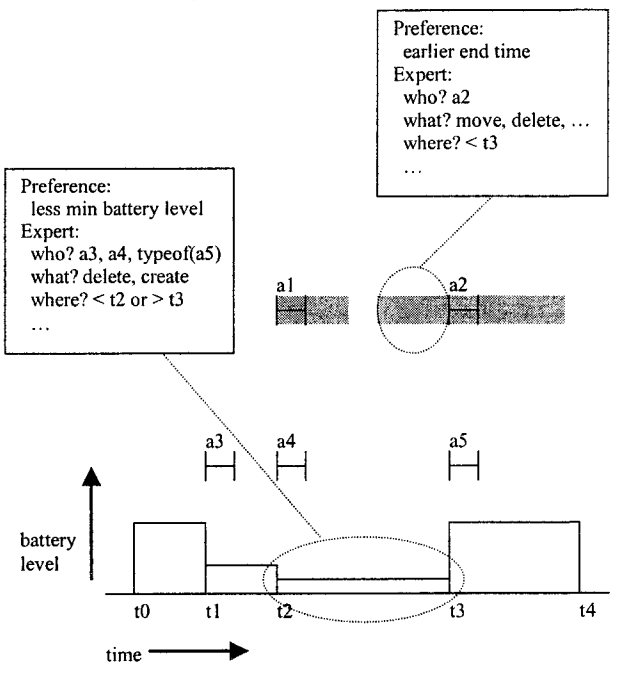

Figure 2: Local improvement experts.

higher minimum resource value, activities that decrease the resource during this time can be moved away from the minimum value. In Figure 2, activities a3 and a4 both contribute to the low minimum battery level. If the preference is for a lower maximum resource value, activities that increase the resource during this time can be moved away from the maximum value. Similar (but probably less useful) cases exist for higher maximum and lower minimum resource values. Moving an activity does not significantly change the average resource value and therefore is not considered for preferences on averages.

Resource/state change count expert. A simpler class of expert is used for improving scores of preferences on the number of times a resource or state variable changes over time. Adding activities that use the resource or state variable will increase the number of changes. Deleting will decreases the number of changes. Because each activity makes a constant number of changes on a resource/state variable, moving has no impact on the change count.

State duration expert. The last class of improvement expert works on state duration preferences. Activities that change the state variable can be created, deleted, or moved in order to change the amount of time spent in a particular state. When the preference is for a longer duration, activities that change to the specified state can be created at times when the variable is in a different state. Conversely, when the preference is for a shorter duration, activities that change to any other state can be created at times when the variable is in the specified state. For example, an activity that switches an instrument off can shorten times where the instrument has been left on. Similar reasoning is used when deleting or moving activities.

Monotonic Preference Assumption

In order to make improvement calculations tractable, we make a *monotonic preference assumption*, requiring each mapping from plan variable to quality metric to either be consistently increasing or decreasing within a given range of the variable. For preferences centered on a value, the score must increase for values less than the specified center value, and decrease for values greater than the center value. In this way, the problem can be restated as simply identifying modifications that increase or decrease the current values of plan variables participating in preferences. For example, if a variable with integer domain [1,10] and current value 4 has a decreasing preference, then only values in the range [1,3] will increase the score for this preference.

Iterative Optimization

The full set of potential plan improvements can be quite large. Once the automated expert has identified this set, we search for more optimal plans by iteratively selecting and making improvements (see Figure 3). We call this technique *iterative optimization* because of its similarity to iterative repair techniques used for repairing plan conflicts (i.e., constraint violations). More specifically, the iterative optimization algorithm first selects a preference from the list of sub-optimal (i.e., score < 1) preferences. Typical heuristics for this decision include selecting a preference with one of the lowest scores or one with the most potential gain (weight * (1 – score)). Next, the algorithm must decide which type of modification to perform for the selected preference.

We allow several types of plan modifications in ASPEN. New activities can be instantiated from types defined in the domain, scheduled activities can be moved to different time or simply deleted, and local variables in activities can be changed. After making a local improvement, the resulting plan may not be optimal. The iterative optimization algorithm continues by selecting another preference, and repeating the improvement process. After each improvement, the resulting overall score is compared with the best score achieved so far. If the current score exceeds the best score, the current plan is saved. The algorithm halts when the maximum score is attained, or when a specified time limit is reached. If an optimal plan was not found, the saved plan with the best score is returned.

```
Iterative Optimize (T)
    Let P = P_best  = current plan
    Let S = S_best  = current score
    While (S <1 and time < T)
        Let Q = set of preferences with sub-optimal score
        q = choose(Q)
        M = E_q(P) //get the set of modifications
        m = choose(M)
        P = m(P) // apply the chosen modification
        S = score(P)
        If (S > S_best) // save if best-so-far
            S_best = S
            P_best = P
    Return P_best
```

Figure 3: The ASPEN optimization algorithm. $E_q(P)$ returns the set of modifications for plan P calculated by the expert E for improving preference q.

When making modifications during iterative optimization, a few simple, domain-independent heuristics are used to avoid violating hard constraints. However, adhering to the plan constraints may be too restrictive, precluding modifications necessary for improving quality. Therefore, the iterative optimization algorithm may create conflicts while searching for an optimal plan. Because it is unknown how the plan will change to achieve feasibility, we do not attempt to define quality for inconsistent plans. The iterative repair algorithm is invoked to restore feasibility before continuing with optimization.

The iterative optimization algorithm does not perform strict hill-climbing. Since modifications are applied to increase the score for a single preference, scores for other preferences may have suffered and the overall score for the plan may have decreased after a single iteration. This suggests that a subset of the preferences represent competing objectives. Although we focus on a single preference at each step in optimization, we do not necessarily maximize the preference score. We only attempt to increase the score by stochastically choosing one of the potential improvements. Therefore, we would expect competing preferences with a large disparity to eventually reach a compromise rather than thrash between a high score for one and a low score for the other.

CONTINUOUS IMPROVEMENTS

One major advantage to a local, iterative algorithm is that it is naturally capable of running on any plan at any time. Because the algorithm is stateless, arbitrary modifications can be made between runs. In addition to changes from user interaction, modifications may come from updates reported by execution monitoring.

The CASPER system (Chien et al. 1999) was developed to continuously initiate and monitor the execution of an ASPEN plan. During execution we may notice differences between actual and expected values for activities or resources. Simply put, things might

not always go as planned. When this happens, we need to update the plan to reflect the actual values. One possible outcome from updates might include the generation of new plan conflicts that need to be repaired before further execution. For this purpose, CASPER utilizes the iterative repair algorithm at necessary times during execution.

Another possibility, however, is that the execution updates suggest new opportunities that were previously unavailable. In this case, the new plan has room for improvement. To handle these situations, we have incorporated the iterative optimization algorithm into CASPER as well. As the result of a plan update, CASPER may attempt one of two things: repair any new conflicts or improve the scores of preferences. One example of a new opportunity might be the early completion of an activity. If activities finish earlier than expected (including failures), the plan can continuously be improved by shifting future activities to an earlier time. Eventually, enough space may become available to achieve additional goals.

CASE STUDY

New Millennium Earth Observer 1 (EO-1) is an earth imaging satellite featuring an advanced multi-spectral imaging device. EO-1 mission operations consists of managing spacecraft operability constraints (power, thermal, pointing, buffers, consumables, telecomm, etc.) and science goals (imaging surface targets within specific observation parameters). One of the interesting constraints involves the Solar Array Drive (SAD) which keeps the solar arrays facing the sun. For a few minutes before and during each data-take, the SAD must be locked to avoid spacecraft jitter, which can corrupt data. The EO-1 model consists of 14 resources, 10 state variables and total of 38 different activity types.

The EO-1 model includes preferences for: more science goals, more time with the SAD tracking the sun, fewer changes of the SAD state, and less deviation from the preferred separation of data-take and SAD locking activities. The last preference has a high score centered on a value because if the settling time is too small there will be too much jitter, but if the separation is too large the solar array power output will suffer.

Table 2 gives the results for the average values for each of the preference variables in the best EO-1 plan. The second row gives the number of hours planned for the SAD in the "tracking" state while the third row gives the number of SAD operations in the plan. In the best case, the SAD would simply track the sun 24 hours a day. However, the observations require a small amount of time with the SAD locked, which requires one operation to lock the SAD and one to return it to tracking mode. The last row contains the average number of minutes between each SAD lock activity and the subsequent data-take. The desired separation is five minutes.

RELATED WORK

Much of the recent work in plan optimization has been looking at ways to integrate linear programming (LP) techniques with symbolic AI and constraint propagation (Baptiste, Le Pape, and Nuijten 1995; Hooker et al. 1999; Kautz and Walser 1999; Vossen et al. 1999). While IP formulations have the advantage of taking a global view of plan quality, they can be difficult to develop and computationally expensive to solve when including representations for state, resource, and temporal constraints. PYRRHUS (Williamson and

Hanks 1994) is a partial-order planner that must evaluate the utility of partial-plans in order to address optimization. In order to compute the upper bound on utility of partial plans, they must make a more restrictive assumption that *overall* quality does not decrease when making refinements. (Myers and Lee 1999) view the optimization problem as providing a set of qualitatively different plans, which can then be further refined by human planners. The CABINS (Miyashita and Sycara 1995) system uses a similar iterative optimization algorithm to improve complete but sub-optimal schedules. Here, case-based reasoning (CBR) is used to learn preferences from the user's evaluation of the plans. Finally, our approach is a specialization of black-box optimization techniques. While black-box algorithms are generic and can optimize arbitrary quality functions, the large search space makes both finding and applying the appropriate technique prohibitively expensive.

CONCLUSIONS

We have described an approach for representing and optimizing user quality metrics (i.e., soft constraints) using generic preferences for values of arbitrary variables in the plan. In our approach, the set of local improvements can be efficiently computed for each preference in a domain-independent fashion. To accomplish this, the representation is restricted to monotonic functions for mapping plan values to quality metrics. We have demonstrated the feasibility of our approach in a spacecraft operations domain.

ACKNOWLEDGEMENTS

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

Baptiste, P.; Le Pape, C.; and Nuijten, W. 1995. Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling, In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research.*

Cesta, A.; Oddi, A.; and Smith, S. F. 1999. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows, In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (IJCAI-99), 1022-1028.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999. Integrated Planning and Execution for Autonomous Spacecraft, In *Proceedings of the 1999 IEEE Aerospace Conference.*

Chien, S.; Smith, B.; Rabideau, G.; Muscettola, N.; and Rajan, K. 1998. Automated Planning and Scheduling for Goal-Based Autonomous Spacecraft, *IEEE Intelligent Systems*, September/October, 50-55.

Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D.; 1997. Toward an Application Framework for Automated Planning and Scheduling, In *Proceedings of the 1997 International Symposium of Artificial Intelligence, Robotics and Automation for Space* (iSAIRAS-97), Tokyo, Japan.

Hooker, J. N.; Ottosson, G.; Thorsteinsson, E. S.; and Kim, H. 1999. On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization, In

Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), 136-142. AAAI Press.

Kautz, H.; and Walser, J. 1999. State-space Planning by Integer Optimization, In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (AAAI-99), 526-533. AAAI Press.

Minton, S.; and Johnston, M. 1988. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, *Artificial Intelligence*, 58:161-205.

Miyashita, K.; and Sycara, K. 1995. CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair, *Artificial Intelligence*, 76(1-2):377-426.

Myers, K. L.; and Lee, T. J. 1999. Generating Qualitatively Difference Plans through Metatheoretic Biases, In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (AAAI-99), 570-576. AAAI Press.

Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations Using the ASPEN System, In *Proceedings of the 1999 International Symposium of Artificial Intelligence, Robotics and Automation for Space* (iSAIRAS-99).

Smith, B.; Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1998. Representing Spacecraft Mission Planning Knowledge in ASPEN, *Artificial Intelligence Planning Systems Workshop on Knowledge Acquisition*, Pittsburgh, PA.

Smith, S. 1994. OPIS: An Architecture and Methodology for Reactive Scheduling, In *Intelligent Scheduling*, Morgan Kaufman, San Francisco, CA.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the Use of Integer Programming Models in AI Planning, In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (IJCAI-99).

Williamson, M.; and Hanks, S. 1994. Optimal Planning with a Goal-Directed Utility Model, In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (AIPS-94), 176-181. AAAI Press.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair, In *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, CA. 241-256.